(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 777 178 A1

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
04.06.1997 Bulletin 1997/23

(51) Int. Cl.^{6}: **G06F 9/44**

(21) Application number: **96308493.4**

(22) Date of filing: **25.11.1996**

(84) Designated Contracting States:
DE FR GB

(30) Priority: **30.11.1995 US 565374**

(71) Applicant: **INTERNATIONAL BUSINESS MACHINES CORPORATION**
Armonk, NY 10504 (US)

(72) Inventor: **Coskun, Nurcan**
Austin, Texas 78727 (US)

(74) Representative: **Waldner, Philip**
IBM United Kingdom Limited,
Intellectual Property Department,
Hursley Park
Winchester, Hampshire SO21 2JN (GB)

(54) Data processing system

(57) Providing dynamic roles for objects in an object-oriented programming environment. A mechanism adds roles dynamically for an object depending on the context of the object. The IBM System Object Model (SOM) is the principal building block for providing this function. The dispatch resolution of the SOM is used to provide the functions of this invention the "somDispatch" methods feature. The "somDispatch" methods is overridden in new classes to implement application specific dispatch mechanism. Dynamic objects are represented as lists of objects where objects are added to the list at any time. For example, when a program is started, a person object will only have a student object. Before calling a function requiring teacher characteristics, the teacher role is added to the person object. When the function call returns, the teacher role is deleted from the person object if the role is no longer needed.

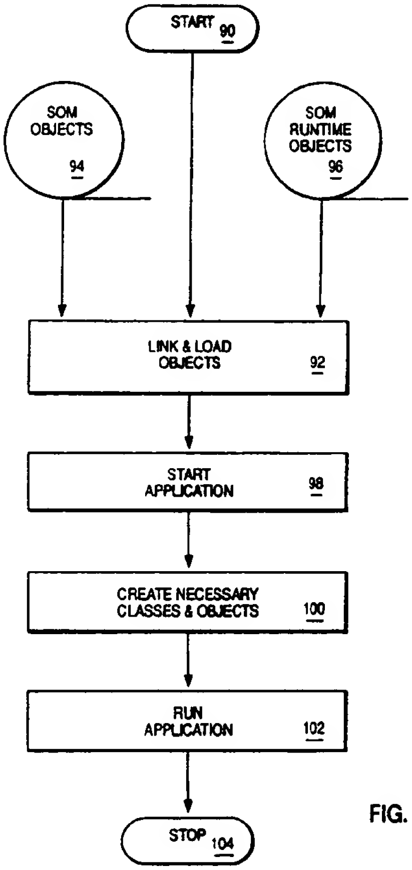

FIG. 9

EP 0 777 178 A1

Description

The present invention relates to data processing systems, and more particularly, to providing dynamic objects for object-oriented programming applications in a System Object Model (SOM).

After years of debate and discussion, object-oriented programming languages (or OOP) are becoming mainstream programming technologies. The OOP languages offer improved programmer productivity, reusable code, and easier maintenance. A central aspect of object-oriented programming models is called method resolution. This is the mechanism used to select a particular method given an object, the method's ID, and the arguments to the method. In some prior art object models, method resolution consists of determining an offset into an object's specific table of procedure entry points based on an analysis of the program's source code. These objects are referred to as static objects. Static objects do not allow a choice of which method to select during program execution. Static objects expect the choice of which method to execute for a particular method's ID, and a particular object to remain constant once program execution begins. While most static models allow the resolution to be preformed once, the selected method may be called repeatedly without going through resolution as long as the same object is to be used.

Another prior art object-oriented programming language offers a dynamic model which consists of using the name of the object to determine a specific method at runtime. In dynamic models, the choice of which method to select can change during the time of a program's execution. Unlike static objects, which allow a selected method for the same object to be called repeatedly without going through resolution, once resolution has occurred, dynamic models cannot perform such a procedure because the set of methods that a class overrides from it ancestor classes can change during a program's execution, thus changing the method that should result from resolution with a particular method's ID, and a particular object while the program is running.

In any case, static or dynamic objects must be capable of corresponding to different roles. For example, a person could represent a student, teacher, father, etc. In order to make the same entity have different responsibilities in different contexts, the prior art has use the object-oriented mechanism of multiple inheritance. Multiple inheritance represent objects as objects that inherits from different classes. Thus, objects are required to carry the overhead of their complexity in every context.

It is desirable to have a mechanism that can add roles to objects dynamically, depending on the context of the object while limiting the overhead requirements.

There is disclosed a method and apparatus for providing dynamic roles for objects in an object-oriented programming environment. A mechanism is provided that permits roles to be dynamically added for an object, depending on the context of the object which results in efficient programs. The IBM System Object Model (SOM) is the principal building block for providing this function. The IBM System Object Model implements three types of method resolution. The first is offset resolution which consists of determining an offset into an object specific table of procedure entry points based on an analysis of the program's source code. The second is dynamic resolution which consists of using the name of the object to determine a specific method at runtime. The third is dispatch resolution which is used to provide the functions of this invention. In dispatch resolution, method resolution is decided by the implementation of "somDispatch" methods. This invention overrides "somDispatch" methods in new classes to implement application specific dispatch mechanism. Dynamic objects are represented as lists of objects. Objects are added to the list at any time. For example, when a program is started, a person object will only have a student object. Before calling a function requiring teacher characteristics, a teacher role is added to the person object. When the function call returns, the teacher role is deleted from the person object if the role is no longer needed.

In summary, dynamic roles objects are associated with a role list object. Methods are provided to add a role to an object or delete a role. This is achieved by overriding the somDISPATCH method. The procedure searches for all objects in the role object list to find the first one having the needed method.

According to a first aspect of the invention, there is provided a method, implemented in a computer system, for creating a dynamic object in an object-oriented environment, comprising the steps of: creating a role list object having at least one dynamic object in said computer system in said object-oriented environment; and providing methods in said object-oriented environment to add a role to said dynamic object.

Preferably the method further comprises the steps of: determining that said role is required in an application program in said computer system in said object-oriented environment; searching said role list object in said computer system for said dynamic object containing said role.

According to a second aspect of the invention, there is provided an apparatus for creating a dynamic object in an object-oriented environment, comprising: means for creating a role list object having at least one dynamic object in said computer system in said object-oriented environment; and means for providing methods in said object-oriented environment to add a role to said dynamic object.

According to a third aspect of the invention there is provided a method implemented in a computer system for accessing a dynamic object in an object oriented environment, comprising: creating a list object in said object oriented environment; and creating a dynamic role object in said computer system in said object oriented environment having a plurality of roles as a subclass of said list object.

The invention will now be described, by way of example only, with reference to the accompanying drawings, in which:

Figure 1 is a block diagram representation of a computer/workstation where the invention may be practiced;

Figure 2 is an object in an object-oriented environment;

Figure 3 is a block diagram of a System Object Model (SOM) data structure in an object-oriented environment;

Figures 4-8 are block diagrams of a list object with dynamic role objects of this invention; and

Figure 9 is a flow diagram of an application using the dynamic role objects of the invention.

The embodiments of the invention provide a method and apparatus for creating dynamic roles for objects in an object-oriented programming environment. The embodiments are preferably practiced in the context of an operating system resident on an IBM PS/2 computer available from the IBM Corporation. A representative hardware environment is depicted in Figure 1, which illustrates a typical hardware configuration of a workstation in accordance with the subject invention having a central processing unit 10, such as a conventional microprocessor, and a number of other units interconnected via a system bus 12. The workstation shown in Figure 1 includes a Random Access Memory (RAM) 14, Read Only Memory (ROM) 16, an I/O adapter 18, for connecting peripheral devices such as disk unit 20 to the bus, a user interface adapter 22 for connecting a keyboard 24, a mouse 26, a speaker 28, a microphone 32, and/or other user interface devices such as a touch screen device (not shown) to the bus, a communication adapter 34 for connecting the workstation to a data processing network and a display adapter 36, for connecting the bus to a display device 38. The workstation, in the preferred embodiment, has resident thereon the OS/2 operating system, and the computer software making up this invention which is included as a toolkit.

With reference to Figure 2, a diagram of objects in an object oriented system is depicted in accordance with a preferred embodiment of the present invention. An object encapsulates data and the methods needed to operate on that data. Objects can be represented by a "doughnut diagram" such as shown in Figure 2. Object data 42 is shown in the centre of the doughnut surrounded by the applicable methods 44 through 54. Data 42, may be modified only by the methods of that object. Methods 44-54 are invoked by receiving messages from other objects. A typical object oriented system will have a message router 56, that routes messages between objects. Thus, object 58 causes Method 48, to be invoked by sending a message 55, to message router 56, that in turn sends message 53, to Method 48 of object 40. An object, as used in this invention, has the properties of encapsulation, inheritance, and polymorphism. Encapsulation refers to the hiding of an object's implementation details. Inheritance refers to a technique of specifying the shape and behaviour of a class of objects, called a derived class, or a subclass, as incremental differences from another class, called a parent class or superclass. Polymorphism refers to the ability to hide different implementations behind a common interface, simplifying the communications among the objects.

Objects are grouped into classes of related objects. The class description contains information relevant to all objects in a class, including a description of instance variables maintained by each of the objects and the available object methods. An object instance is created (or "instantiated"), based on that information and has the properties defined in the object class. For example, the object class DOG, can include the instance variables "dog_type", and "dog_name", and a "bark" method implementing the response to a bark message. An instance of dog, e.g., Rover, will maintain the type and name instance variables for itself and will respond to the bark message.

Abstract classes are used to describe the interfaces and methods expected to be used by a class without providing details on the implementation of those methods. Abstract classes are useful where the implementation details are to be left to the implementor. Concrete classes are created as subclasses of abstract classes and implement those classes.

With reference to Figure 3, there is shown a basic SOM data structure. A state data structure 60, for a particular object is shown where the first full word 62, contains the address of the object's method procedure table 66. The method procedure table 66, contains the address of the class object data structure 68, and addresses of various methods for the particular object 72 and 74. The address 68, points to the class object data structure 70. All objects that are of the same class as this object also contain an address that points to this method procedure table 66. Any methods inherited by the objects will have their method procedure addresses at the same offset in memory as they appear in the method procedure table 66, of the ancestor class. Addresses of the blocks of computer memory containing the series of instructions for two of the method procedures are set forth at 72 and 74. Locations 76 and 78, are in computer memory and contains the series of instructions of a particular method procedures pointed to by the addresses at 72 and 74. A complete description of the IBM System Object Model can be found in IBM docket AT9-91-072, Serial Number 07/805,778, U. S. Patent Number 5,421,016, entitled "System and Method for Dynamically Invoking Object Methods from an Application Designed for Static Method Invocation", which is incorporated herein by reference.

Turning now to Figure 4, there is shown a block diagram using a list to add/delete roles to a dynamic object. The list object 80, contains pointers to SOM objects representing a person object 82, student object 84, and a teacher object 86. The invention describes a technique that can add roles to the object dynamically, depending on the context. The person object 82 may contain information on a particular individual including the birth date, birth place, name, etc. The student object 84, contains specific information on a student including grade point average (GPA), major, grade for a particular class, etc. The teacher object 86, contains particular information on a teacher including salary, specialization, students in class, etc. One skilled in the art will appreciate that any number of dynamic objects with various roles can be added to the list object 80.

Turning now to Figures 5-8, the technique for accessing the list object containing the dynamic role objects will be described. When an application program is started, for example, which requires all the people born in Austin, Texas, only the person object 82 is required. As shown in Figure 5, the list object 80, will be searched and only the person object will be loaded. Likewise, if the application program is required to calculate all the students with a GPA of 4.0, the list object 80, will be searched as shown in Figure 6, and only the student object 84, will be loaded. In the same way, an application program that seeks all teachers teaching Math, will search the list object 80, in Figure 7, and load only the teacher object 86. All of the dynamic object will be loaded when an application is called upon to find all the teachers who are teaching Math, and who are also taking a history class, and who are younger then 50 years old. As shown in Figure 8, the list object 80, will be searched and the person object 82, student object 84, and teacher object 86, will be loaded. Returning to Figure 4, it will be apparent to those skilled in the art that given a list object 80, additional dynamic objects (e.g., 82, 84, 86) containing roles can be added to the list object 80, at any time. It will also be apparent to those skilled in the art that when an application program is started requiring only the person object 82, and later requires teacher characteristics, the teacher object 86, may be loaded, and when the function call returns, the teacher object 86, may be deleted if no longer needed.

Turning to Figure 9, there is shown a flow diagram for executing the dynamic role objects of the invention. The invention allows an application to invoke the dynamic role objects. Dynamic invocation refers to the ability of a program to determine the method to call at run time and not when the application is compiled. Dynamic resolution systems require run time information to accurately resolve the method call. As illustrated in Figure 9, an application using the SOM dynamic role objects of the invention commences processing at block 90 and proceeds to block 92. At block 92, the procedure performs a dynamic link and load of the dynamic SOM objects 94, based on the list object of Figure 5, and the SOM run time library 96. At block 98, the application is started, which invokes the creation of necessary classes and objects as set forth in function block 100. Finally, the application is executed as shown in function block 102 and control is terminated at terminal block 104.

In summary, a data processing system provides dynamic roles for objects in an object-oriented programming environment. A mechanism adds roles dynamically for an object depending on the context of the object. The IBM System Object Model (SOM) is the principal building block for providing this function. The dispatch resolution of the SOM is used to provide the functions of this invention the "somDispatch" methods feature. The "somDispatch" methods is overridden in new classes to implement application specific dispatch mechanism. Dynamic objects are represented as lists of objects where objects are added to the list at any time. For example, when a program is started, a person object will only have a student object. Before calling a function requiring teacher characteristics, the teacher role is added to the person object. When the function call returns, the teacher role is deleted from the person object if the role is no longer needed.

While the invention has been described with respect to a preferred embodiment thereof, it will be understood by those skilled in the art that various changes in detail may be made therein without departing form the spirit, scope, and teaching of the invention. Accordingly, the herein disclosed invention is to be limited only as specified in the following claims.

Claims

1. A method, implemented in a computer system, for creating a dynamic object in an object-oriented environment, comprising the steps of:

   creating a role list object having at least one dynamic object in said computer system in said object-oriented environment; and

   providing methods in said object-oriented environment to add a role to said dynamic object.

2. The method of claim 1, further comprising the steps of:

   determining that said role is required in an application program in said computer system in said object-oriented environment;

   searching said role list object in said computer system for said dynamic object containing said role.

3. An apparatus for creating a dynamic object in an object-oriented environment, comprising:

means for creating a role list object having at least one dynamic object in said computer system in said object-oriented environment; and

means for providing methods in said object-oriented environment to add a role to said dynamic object.

4. The apparatus of claim 3, further comprising:

means for determining that said role is required in an application program in said computer system in said object-oriented environment;

means for searching said role list object in said computer system for said dynamic object containing said role.

5. A method implemented in a computer system for accessing a dynamic object in an object oriented environment, comprising:

creating a list object in said object oriented environment; and
creating a dynamic role object in said computer system in said object oriented environment having a plurality of roles as a subclass of said list object.

6. The method of claim 5 further comprising:

determining that an application program in said object oriented environment requires an indicated one of said plurality of roles in said dynamic role object; and

searching said list object for said indicated one of said plurality of roles in said role object.

7. The method of claim 6 including the step of determining e global storage setting in a process storage area.

8. The method of claim 7 wherein the step of accessing said user's credentials includes the step of retrieving said user's credentials from said process area, if present.

9. The method of claim 8 wherein the step of accessing said user's credentials includes the step of retrieving said user's credentials from said local storage area when said user's credentials is not in said process storage area.

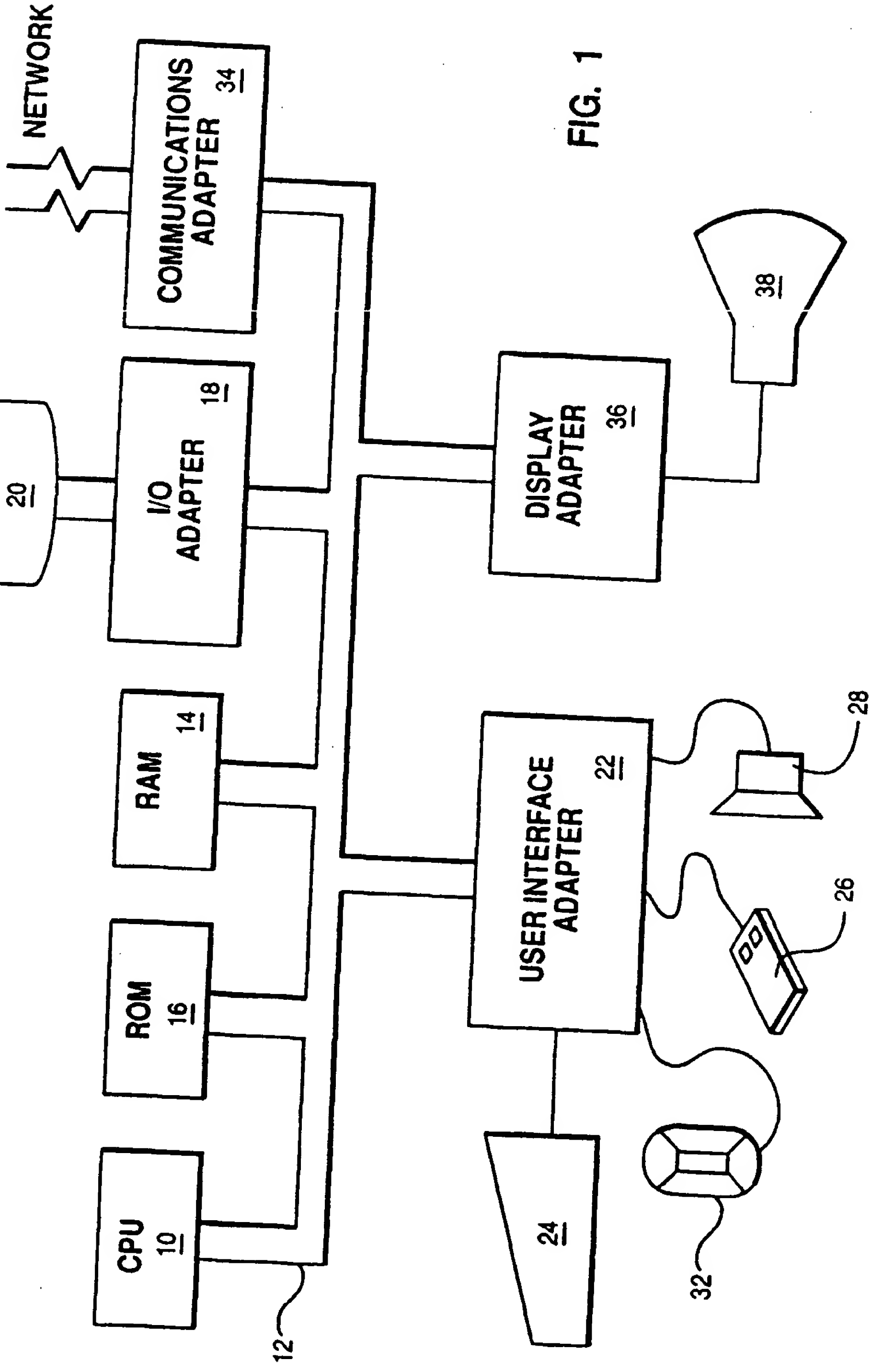
NETWORK
COMMUNICATIONS ADAPTER 34
I/O ADAPTER 18
20
RAM 14
ROM 16
CPU 10
12
DISPLAY ADAPTER 36
38
USER INTERFACE ADAPTER 22
28
26
24
32

FIG. 1

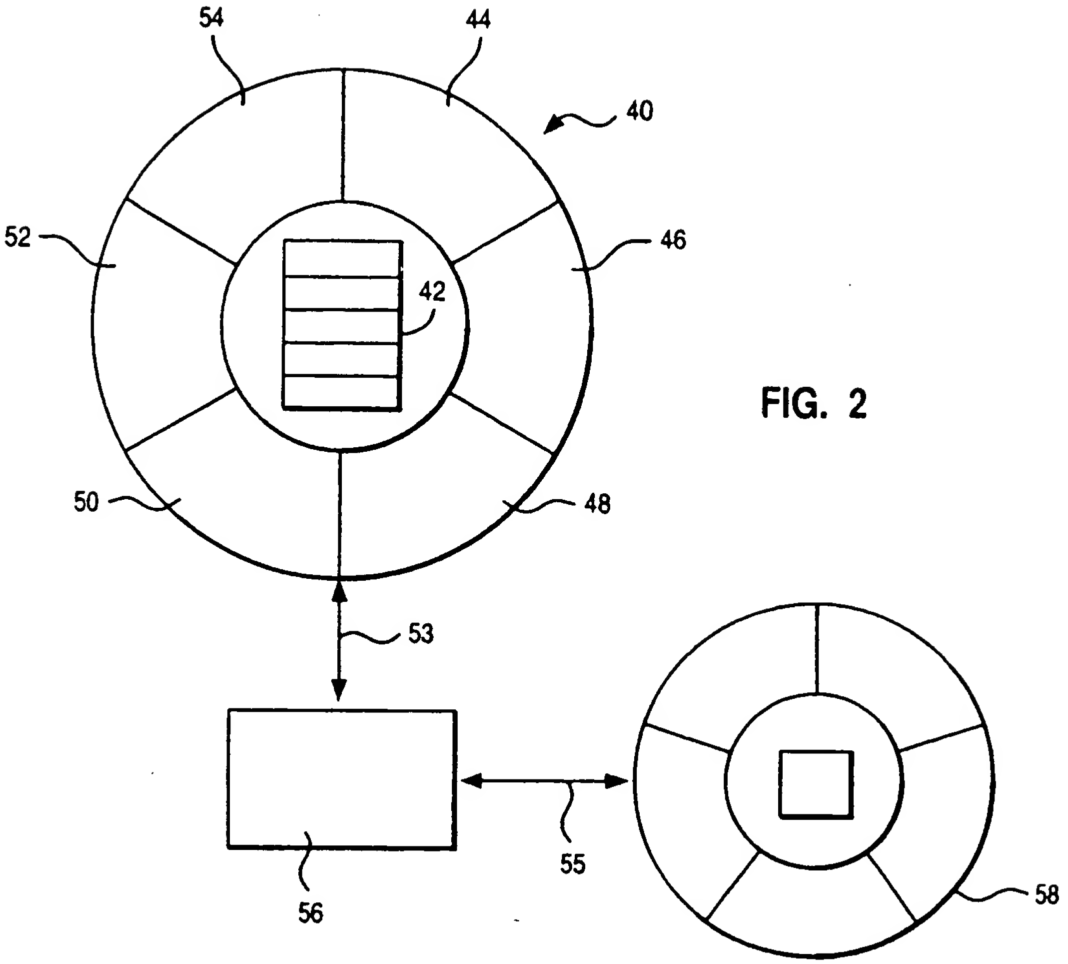

FIG. 2

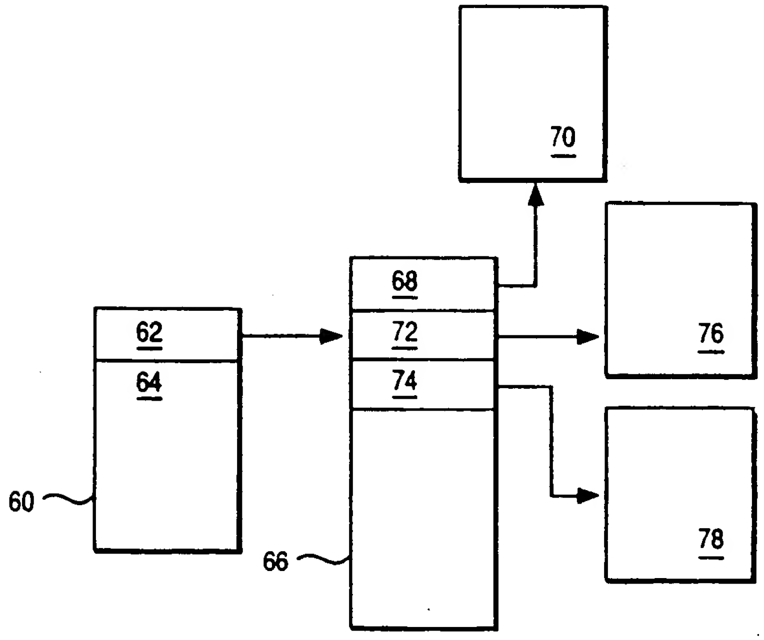

FIG. 3

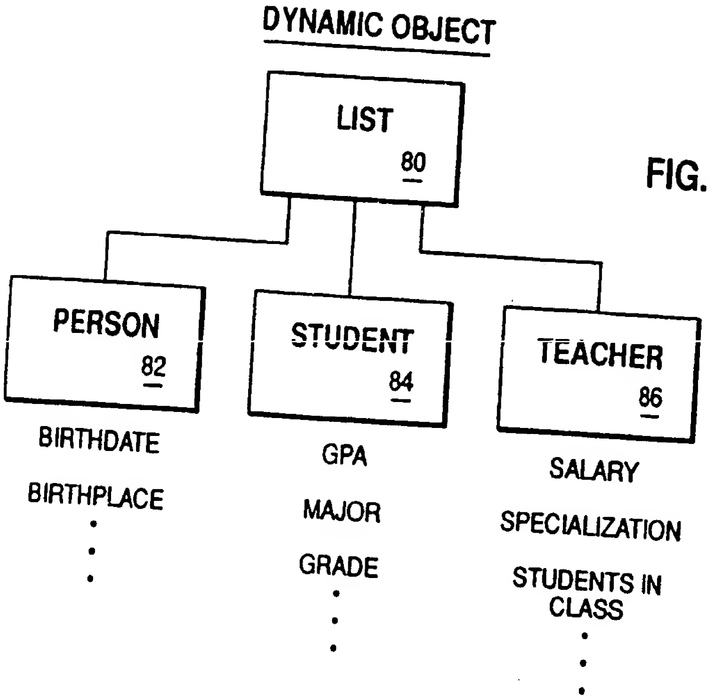

FIG. 4

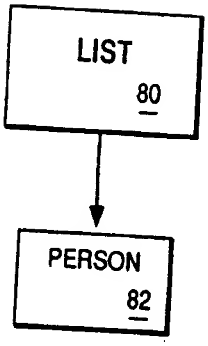

FIG. 5

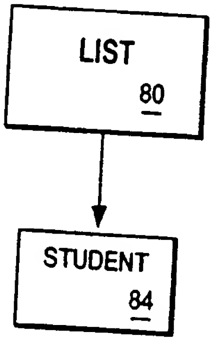

FIG. 6

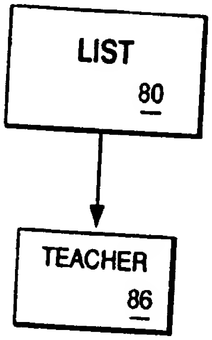

FIG. 7

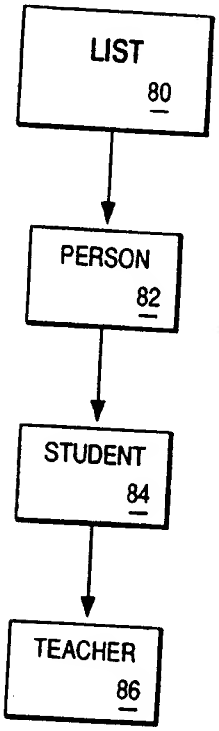

FIG. 8

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 96 30 8493

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
D,A	US 5 421 016 A (CONNER ET AL.) 30 May 1995 * the whole document * ---	1-6	G06F9/44
A	IBM OS/2 DEVELOPER, vol. 5, no. 2, US, pages 94-102, XP002026680 ROGER SESSION AND NURCAN COSKUN: "Method Resolution in SOM" Spring 1993 * page 100, left-hand column, last paragraph - page 102, left-hand column, last line * -----	1-6	
			TECHNICAL FIELDS SEARCHED (Int.Cl.6)
			G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	28 February 1997	Fonderson, A

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

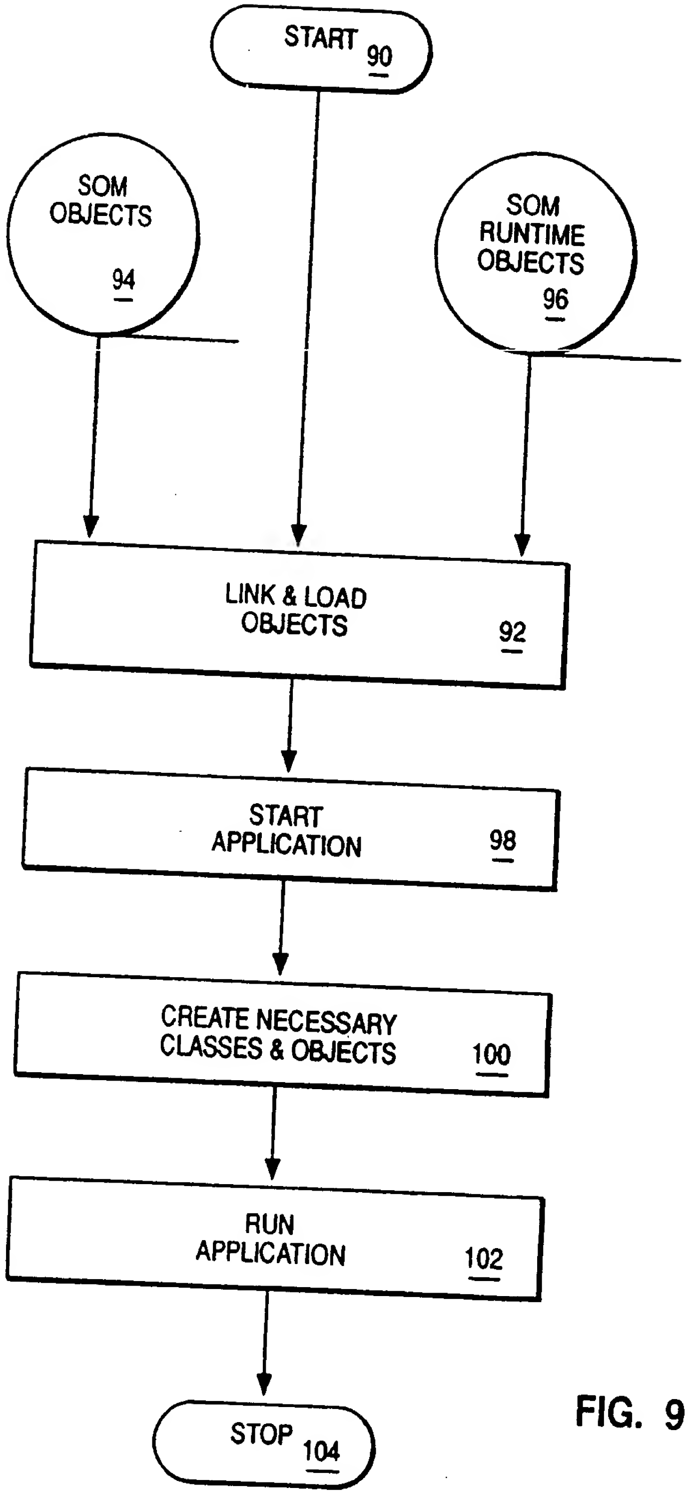
START 90
SOM OBJECTS 94
SOM RUNTIME OBJECTS 96
LINK & LOAD OBJECTS 92
START APPLICATION 98
CREATE NECESSARY CLASSES & OBJECTS 100
RUN APPLICATION 102
STOP 104

FIG. 9